SİSTEM PROGRAMLAMA

Yrd. Doç. Dr. Hüseyin Pehlivan

username: merve

password: 911abc

26 02 2007
1

Ders konusu:

A) Unix Sistemlerde Script Yazılım

B) Sistem Çağrılarıyla Problem Yazılım

(1) Unix

Bir işletim sistemidir. Kullanıcı ile bilgisayar arasındaki kuralları izmaridir. Bilgisayar kaynaklarını yönetir (files, programs, disks, network)

Modern işletim sistemidir. Kararlı, yapılandırılabilir, çoklu kullanıma açık.

Birçok bilimsel ve endüstriyel alanda kullanılır, kaynak kodu açıktır, programlama çevresi gelişmiş. Ücretsiz sahip olunabilir.

Dünyada %65 Unix / Linux kullanılıyor.

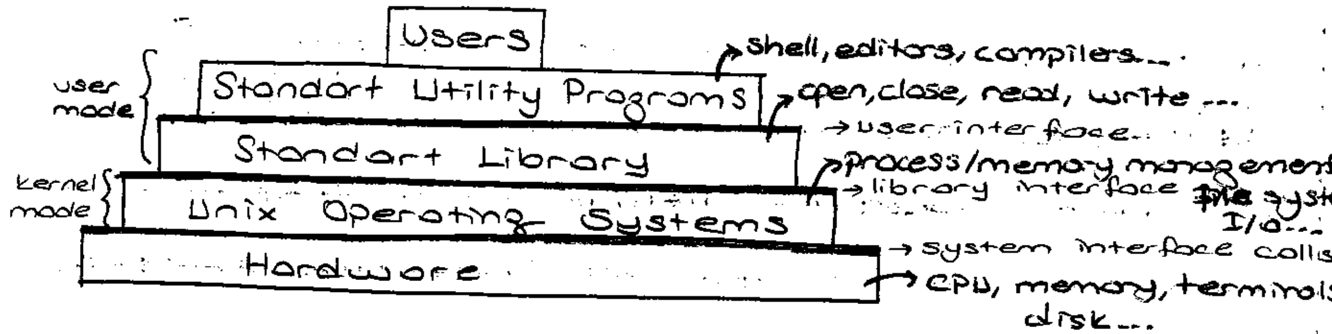
Unix 1970'lerde Bell Lab'larda Ken Thompson ve Dennis Richie tarafından programcılar için geliştirilmiş.

Ken B'de program yazıyor, B yetersiz kalınca Dennis B'den C dilini üretip UNIX'i bununla yazıyor. %95'i C, %5'i (kernel) assembly ile yazılmıştır.

BSN → Unix sisteminin 2 ayrı koldan geliştiği, başlangıçta tek bir sistem olduğunu gösteriyor. Ör: SunOS 4, Ultrix

SYS4 → ÖR:

Unix Tabanlı Sistemde Tabakalar:



fopen dersek library interface'i,
open dersek sistem call interface'ini kullanmış oluruz. İki emri de aynı kod içerisinde kullanabiliriz.

Kernel: Sistem donanımını kontrol eden ve çeşitli alçak seviyeli işlemleri kontrol eden bir yazılımdır.

İstek kernela bildirilir, kernel servisleri tahsis eder.

Shell: kullanıcı seviyeli programdır. Kullanıcıdan aldığı komutları bildirir, komutların icrası başlatır ve sonlandırır, asıl görevi budur. Komutu kernel koşar.

UNIX'te 200'ü aşkın utility programları ya da komutları bulunur. Bunlar OS'nin parçası değildir, yine de kullanıcının kullanımına uygun komut-

lardır (dosya işlemleri, hesaplamalar ve yazılım geliştirme araçları gibi). Uzaktan UNIX'e bağlanıldığında bize bir terminal sunulur. UNIX sistemine login yapmak (bağlanmak) bir accounta (hesaba) ihtiyaç duyar. Hesap kullanıcı ismi ve şifresi ile ilişkilendirilmiştir.

Login: init ile karşılanırlar. init OS çalıştırıldığında ilk olarak oluşturulan süreçtir. PID (process ID)'si 1'dir. init bir login işlemiyle karşılaştığında bir child süreç oluşturulur. getty programı user name ister. Alındığında login icra edilir. Login passwordun geçerliliğini kontrol eder. Geçerliyse login shelli çalıştırır.

Kullanıcı tam belirlenmiş shell kullanıcının kullanımına açılır. Login metodu x displayi de kullanabilir. Xdm'e alternatif olarak gdm ve kdm verilebilir. Ama xdm harici sistem yöneticisi tarafında kurulmalıdır. Default değillerdir. xdm otomatik olarak kurulumda çalışan masaüstü programlarını destekleyen birşeydir.

Güvenli Erişim Araçları

Terminal bağlantısı için PuTTY (on Windows)

MindTerm (Java applet)

Masaüstü erişimi için X-Win32 → CDE, KDE, GNOME

Weirdx (Java applet)

Dosya erişimi için



Shell Prompt

$ / % / # / username@hostname> / hostname%

komutları kullanılabilir.

Shell

Kullanıcıların işletim sisteminde program koşa_bilmesini sağlayan programdır.

- who → şu an çalışan programcılar
- date → Sistemin tarihini ve saatini gösterir.
- ls → şu an bulunan dosya ve klasörler
- cat textfile → ?
- ls -l → klasördeki öğeler hakkında daha fazla bilgi

Command Options Argument(s)

Komutun davranışı aynıdır. Optionlar gösterilen bilgiyi komutun davranışının kapsama alanı_nı değiştirir. Örneğin birden çok dosyayı 1 ko_mutla silmek istiyorsak option kullanmalıyız.

Örn ls için ls -l

ls -a

ls -la

ls -a ; ls -l

ls -F

ls al textfile 1

ls al textfile 1 textfile 2

Eğer shell prompt verilmezse demek ki son verilen program hala devam ediyor. İlla prompt istiyorsak CTRL-C ile programın tarası sonlandırılır. CTRL-Z ile programın tarasına ara verilir. Daha sonra kaldığı yerden devam ettirebiliriz.

quit, bye, exit gibi komutlarla da bunu başarabiliriz.

Logging Out

exit ile sistemden çıkılabilir. Kullanıcılar terminale bağlı olarak logout veya CTRL-d de kullanabilir.

(2) UNIX'te Dosya Sistemi

Fiziksel depolama alanına bir interface'dir. Aynı zamanda I/O cihazlarla da bir arayüz oluşturur.

UNIX'te herşey bir dosyadır. UNIX'teki herşeye bir dosyaya erişir gibi erişebiliriz. (Soket, terminal, başka bir makinedeki açık bir dosya, vs. olabilir)

Directory: Diğer dosyaları içeren basit bir dosya. UNIX sistemde, Türkçe harfleri kullanılmaz.

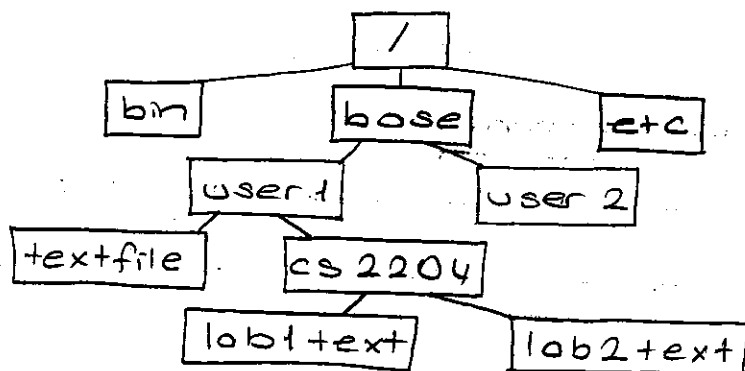
Working directory: Dosya sistemi üzerinde bulunulan o anki konum.

pwd (print working directory): İlk directoryden working directory kadar gelinen yolunu (absolute path - çalışma dizini) yazdırır.

Koşulan herbir komut bulunulan dizini alır. (Eğer başka dizinin ismini parametre olarak vermediysek)

home directory: Sisteme login olunduğun bize sunulan dizi. Home dizin için ~ karakteri kullanılır. (~ user1).

root directory (/):



absolute path → roottan bulunulan yere kadar

Ör: /home / user1 / textfile

~ user1 / textfile

~/textfile

relative path: Bulunulan dizinden başlanarak erişilebilecek yol. • bulunulan dizini, •• bir üst dizini gösterir.

Ör: textfile

CS2204/ lab1txt

Bazı Standart Dizinler

/	Root directory
/bin	Dizinin altındaki standart komutlar (executables)
/dev	Dizin içerisindeki blok ve karakter tabanlı sürücülere linkler
/etc	Hosta özel yapılandırma ve servisler
/home	Kullanıcıların home dizinleri, günümüzde export dizini altına yerleştiriliyor.
/lib	Çeşitli programlama dilleri için library dosyaları
/sbin	Sistem komutları ve utilitiler (boot için gereken)

<u>/tmp</u> Temperary dosyaları (bazı programlar için geçici dosya üretir. Örneğin edit yaparken)

<u>/usr</u> User utilitileri ve uygulamaları /usr/loca

<u>/var</u> Boyutu değişebilen dosyaların yerleştirildiği dizin (system log, spools, emails, yazıcı dosyaları)

Changing Directories

<u>cd</u> - Bulunulan dizini değiştirmeye yarar.

<u>Ör:</u> cd /home/user1

cd /../../user1

ls-lFa 'nın çıktısı

permission	users	group	file size	modified date	file name
↓ file type	↓ number of hard links				

<u>plain (-)</u> → Çoğu dosya, binary veya text.

<u>directory</u> (d) → diğer dosya kümelerini gösteren dosya

<u>link (l)</u> → diğer dosya ve dizinlere pointer

<u>special (b)</u> → blok cihaz (diskler, CD-ROM)

<u>(c)</u> → karakter cihaz (klavye, joystik)

• <u>touch <file></u> ile yeni dosyalar ya da modify yapılır.

- mv <file1> <file2> (1)

  mv <file1> <dir> (2)

  mv <file1> <dir/file2> (3)

mv (move) ile dosyayı hareket ettirir ve ismini değiştirir.

(1) dosya adını
(2) dosya yerini
(3) dosya adını ve yerini
} değiştirir.

- cp <file1> [<file2> | <dir> | <dir/file2>].

cp (copy) kopyalama yapar.

- rm [-i] <file(s)>

rm (remove) dosya(ları) siler. [-i] varsa onay ister.

- mkdir <directory_name>

mkdir (make directory) dizin oluşturur.

- rmdir <directory-name>

rmdir (remove directory) dizin boşsa siler, değilse başarısızdır.

- rm -r <directory-name>

-r (recursively) olursa dizini içindekilerle beraber siler.

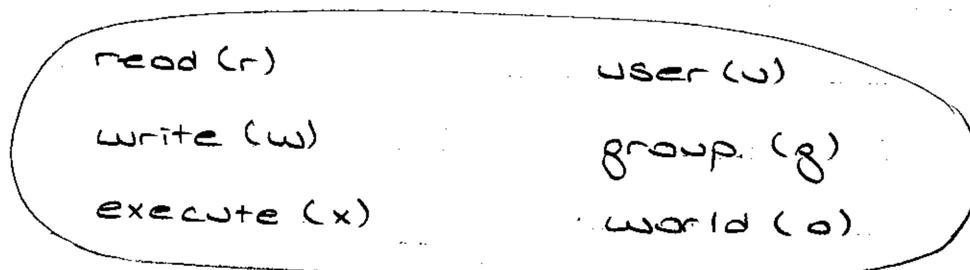
- ln -s <existing-file> <link-name>

ln (link) link oluşturur. -s yoksa hardlink, varsa softlink oluşturur. -s (sembolic link) ile içerisinde yol bulu-

non yeni bir dosya oluşturulur.

File Ownership

Chown



→ permission (izinler)

Her dosya tek bir owner'a sahiptir. Chown owner'ı değiştirmede kullanılır. Genellikle sadece root bu komutu kullanır.

(change mode) chmod <mode> < file(s)>

chmod 700 textfile

rwx	rwx	rwx
1 1 1 = 7	0 0 0 = 0	0 0 0 = 0
user	group	world

chmod g+rw textfile

u g o +/- r w x

group kısmına rw eklenir.
1 1 0 = 6

700 + 60 = 760 octals

File Modification Date

Değişik dizinlerde dosyanın birçok kopyası varsa ya da dosya birkaç kullanıcı tarafından kullanılıyorsa last modification data gerekir.

touch → modification date'i update eder.

cat textfile1 textfile2 → dosya içeriklerini (bir defada) gösterir.
(cat ↓ concatenate)

less / more textfile → dosya içeriklerini gösterir (sayfa sayfa page-down ile)

Wild Cards in File Names

* herhangi bir dosya

? herhangi bir karakter

[] karakter aralığı vermek için ÖR: [a-c]=a,b,c

Other File Systems

- disk-based (harddisk, CDROM, floppy disk)
- distributed (networke dağıtılmış kaynaklar)
- pseudo (memory tabanlı)

③ Text Editting

cp (copy), mv (move), rm (remove), ln (link), cat, less.

Why vi?

Komutlar klavye ile girilir. Text tabanlı bir editör olduğundan mouse kullanılmamalıdır. (ASCII text)

Vi (improved) = Vim

vi Basics

vi [filename(s)]

Windowu kontrol eder. Mode tabanlı bir editördür.

2 modu vardır:

- Command mod
- Insert mod

Genelde komutlar tek karakterlidir ve büyük kü-
çük harf ayrımı vardır.
Modlar arasında geçiş için çeşitli karakterler yazılır.

command → insert   a ya da o
insert → command   ESC

Command Mod

vi editörü her zaman komut moduyla başlar.
Commandlar ① cursor'ü hareket ettirme
② düzeltme
③ dosya operasyonları
④ arama
⑤ insert moda giriş
şeklinde 5 sınıfa ayrılır.

① Cursorü hareket ettirme

← ↑ ↓ →
h j k l

n çoklu ilerlemede kullanılır.

n (j|k|l|h)
↓
sayı

② u son değişikliği geri alır
x karakteri siler
dd satırı siler
dw kelimeyi siler
rx karakterle x'i yer değiştirir.

yy satırı kopyalar.

p kopyalanmış/silinmiş parçaları yapıştırır.

J iki satırı birleştirir.

(3) ZZ yazar ve çıkar.

:wq yazar ve çıkar.

:w yazar

:w filename dosya adını yazar.

:q düzelme'den çık.

:q! kaydetmeden çık.

:e <filename> başka bir dosyayı düzelt.

:n Bir sonraki dosyayı düzelt.

(4) /string yazılan stringi arar.

?string yazılan stringi geriye doğru arar.

n önceki aramanın [illegible] ettirilir

N önceki aramanın tersine arama devam ettirilir.

% (, [, { ile işaretlerini arar.

(5) i kursörün önünden inserte başlanır

I satır başından inserte başlanır

a kursörden sonra appende başlanır

A satır sonundan appende başlanır

o kursörün altında yeni satır oluşturur.

O kursörün üstünde yeni satır oluşturur.

cw kelimeyi yer değiştirir.

C satır sonunu değiştirir.

Chapter_8'de diğer komutlar listelenmiştir.

V harindeki büyük karakterleri yazarken dikkat etmeliyiz.

CTRL-X ile editörden çıkarız.

imax unix sistemlerdeki en yüksek editörlerden biridir. Yazılım geliştirmede tercih edilir. Kullanılan dilin syntaxına göre keywordleri tanıyacak şekilde dedir. Grafik arayüzü bulunur. Bu editörde CTRL-X sonra CTRL-C ile çıkarız.

~$ vim ~$ gvim su

~$ echo $SHELL

~$ echo $DISPLAY

(4) I/Oem Redirection

Three Standart Files

Herbir terminal için standart dosya tanımlanmıştır.

Stdin: Giriş karakter stream } keyboardda

stdout: çıkış karakter stream } terminalde

Stderr: Error mesajı olan yer }

Redirection stdout

Yönlendirmek için > karakteri kullanılır.

>> ise mevcut bir dosyaya sondan itibaren yazma yapar.

$ date > zzz

Bu date komutunun ürettiği çıktıyı zzz'ye yazar. (Baştan itibaren, zzz'nin içeriğini silerek).

more zzz

ile zzz'ye bakarsak bunu görürsüz. Bu komutlardan önce zzz vardıysa üzerine yazar, yoktuysa yeni oluşturur.

$ PWD → çalışma dizininin tam yolunu içerir.

cat file1 >> file2

Redirect Std in

Giriş < karakteri ile yapılır. Bu karakter sayesinde başka devicelardan giriş alabiliriz.

mail user@domain.com < message.txt

at 3am < cmds or at 0945 < cmds

sort < friends > sorted_friends

interactive_program < command_list

Pipes (|)

Bir komutun stdoutunu diğerinin stdin'ine bağlamak.

ls -la | less

ls -al | wc

cat file | wc

Processes (;)

Aynı anda birden fazla program koşar.

date; who. ls -al & wc *

Önce date sonra who komutu icra edilir.

& karakterini önüne alan komut kendinden sonra gelen komutla eşzamanlı olarak koşulur.

Filters

wc kelimelerin/satırların/karakterlerin sayısını verir.

grep verilen paterne (RE) uygun satırları arar.

grep <pattern> <filename>

sort Sıralama yapar.

-r tersten sıralar

-n numerik sıraya göre sıralar.

+2n 2. sütuna göre sıralar.

cut -c1-5 Satırın ilk 5 karakterini gösterir.

-c1,5 Satırın ilk ve 5. karakterlerini gösterir.

-d: -f1,5 /etc/password: user ID'yi name'e map eder

head Dosyanın ilk n satırını görüntüler.

head -n <filename>

tail -n <filename> Dosyanın son n satırını görüntüler.

+n <filename> Dosyanın ilk n satırından sonrasını görüntüler.

diff İki dosyanın farklı olan satırlarını gösterir.

cmp İki dosyanın farklı ilk satırını gösterir.

diff <file1> <file2>
cmp

od Dosyanın içeriğini 8'li tabanda gösterir.

od -c

0000010 → end of file (EOF)'un gösterimi.

ls-lt modification zamanına göre dosyaları listeler.

crypt verilen bir keye göre dosya içerikleri şifrelenir.

crypt key <clear. file> encrypted file

tr girdideki karakterleri verilen parametrelere göre başka bir karakter setine dönüştürür.

tr "[:lower:]" "[:upper:]" < <file>

lower caseleri upper case'e dönüştürür.

uniq Tekrarlanan satırlarda işlem yapar.

uniq -d <file> tekrarlananları yazdırır.

uniq -u <file> tekrarlanmayacak şekilde tekrarlananların sadece 1 tanesini yazdırır.

Process için Komutlar

ps Aktif processleri listele

top Processlerin sistem kullanımını dinamik olarak göster.

kill Processi sonlandır.

time Bir process için zamanlama bilgilerini sakla.

Communication.

talk Sistemdeki, diğer kullanıcılarla mesajlaş-mada kullanılır.
talk smith pts/2

write Başka bir kullanıcıya mesaj göndermede kullanılır.
write smith pts/2
mesg [n/y] : mesajı reddetmek ya da izin vermek.

mail, pine Text tabanlı email programıdır.

ftp, sftp Text tabanlı ftp programıdır,
Şu anda sadece sftp'ye izin veriliyor,
ftp portu kapalı.

telnet, ssh Diğer makinelere direkt bağlantı için.

lynx Text tabanlı web browser

(5) Regular Expressions

grep dosya ve isminde arama

egrep regular expressionları kullanmak için

vi/emacs text editörs

ex satır editörs

sed akış editörs

awk patern tarama dili

perl yazı dili

Temel regular expressionlar içinde bu meta_ karakterler özel anlama sahip değildirler.

?, +, {, }, (,), |

Grep ile beraber kullanılırlar.

\?, \+, \{, \}, \(, \), \|

şeklinde kullanabiliriz.

Biz regular expressionları egrep ile kullanırız.

egrep pattern filename(s)

ÖR: egrep "abc" textfile → abc'yi içeren satırları ekrana basar.

egrep -i "abc" textfile → karakterlerin büyük veya küçük olmasını dikkate olmaz

egrep -v "abc" textfile → abc'yi içermeyen satırları basar.

egrep -n "abc" textfile → satırları numaralı olarak gösterir.

Meta Characters

period (.) tek bir karakteri bildirir.

ÖR: "a.c" → abc, adc, a&c, a;c, ---

"u..x" → unix, uvax, u3(x, ---

asterisk (*) sıfır veya çok sayıda tekrarı olabileceğini gösterir. (Shell'deki wildcard'larla karıştırma)

ÖR: "ab*c" → ac, abc, abbc, abbbc, ...

".*" → 0 veya çok sayıda . → herhangi bir kelime

plus (+) RE'nin bir veya daha çok sayıda olacağını gösterir.

ÖR: "ab+c" → abc, abbc, abbbc, ...

<u>question mark (?)</u> RE'nin 0 veya 1 kez kullanıldığını gösterir.

ÖR: "ab?c" → ac, abc

<u>logical (|)</u> Bundan önce ya da sonra bulunan RE'den ayrı ayrı paternler üretilmesini sağlar.

ÖR: "ab|de" → ab, de

<u>caret (^)</u> Satır başı anlamındadır.

ÖR: ^D.*" → Satırın D ile başlaması gerektiğini söyler.

<u>Dollar sign ($)</u> Satırın sonunu belirtir.

ÖR: ".*d$" → d ile biten satırları gösterir. ".*" olmasa da aynı durdu.

<u>Backslash (/)</u> Kendinden sonra gelenin bir karakter olarak gösterileceğini belirtir.

ÖR: /.txt → .txt

<u>square brackets ([])</u> Üretilen bir patern içinde bulunmasını istediğimiz karakterleri liste olarak girmemizi sağlar.

ÖR: b[a e]k → bak, bek.

parantheses (()) Gruplama yapmak kullanılır.

ÖR: $a(bc)^+$ = abc, abcbc, abcbcbc...

braces ({}) Bir RE'nin kaç kez tekrarlanacağını belirtir.

ÖR: [a-z]{3} → 3 küçük harften oluşan kelimeler.

ÖR: -$ egrep -n "a[^abc]" BASH

Bu örnekte a ile başlayan abc' dışında karakterlerden biri ile devam eden kelimeler ekrana bastırılır.

* Boşluk da özellikle belirtilmiş olabilir. "2 kez" ararken "2kez" yazdırılamaz.

word seaching with egrep

- /usr/dict/words

Bunun içine alfabetik sırada bulunan 5 sesli harf içeren kelimeleri bulmak için RE.

^[^aeiou]*a[^aeiou]*e[^aeiou]

İçinde bütün sesli harflerin göründüğü alfabetik sırada olan kelimeler.

^[^aeiou] = [bcdf...]

* egrep -f ⟶ Komutun, verilen RE expressionun dosyadan okuması gerektiğini söyler.

RE'i alphvowels
↓
bu dosya içinden olması gerektiğini söyler.

egrep -f. monoton ...

6 ya da daha fazla kelimenin alfabetik sırada yer alır, karakterleri alfabetik sırada verirsiz aralarına ? koyarız.

^a?b?c?

grep "..........."

⑥ UNIX Window System

Window sistemleri OS üzerinde bulunan ve kullanıcıya grafiksel arayüz sunan sistemlerdir. (GUI) windows(pencereler), Icons (ikonlar) ve interrupt driven interaction tabanlıdırlar.

Window sistemleri gelişmeden önce Unix sistemleri geliştirilmiş, böyle olduğundan komut satırını etkin kullanmak için UNIX kendi araçlarını geliştirmiştir. Bunlardan biri text tabanlı arayüz sunan SHELL'lerdir. Windowtan önce UNIX text tabanlı kontrolü SHELL'lerle gerçekleştirir. Bu yüzden onlarca SHELL var.

UNIX sistemlerinde window sistemleri, X11 26 gibi. Herhangi bir UNIX sistemi ve Windows sisteminin birlikte kullanılması için X11 26 kütüphanesi ile window uyumlu çalışmalıdır.

X-Server

[illegible] sağlamak için X-server hizmet verir. Bu hem konsol üzerinde, hem de remote olarak çalışma imkanı kullanıcılara verir.

- Donanımla bir interface içerir. (display ...)
- Makina üzerinden de gerçekleştirilebilir.
- Displayi yönetmek için servislere sahiptir.
- Basit grafik işlemlerini destekler. Bazı [illegible] rectangular alanları bölgeleri clientlara tahsis edebilir.
- Client-server mimarisine uygun bir yazım.

The X Client

X client - X server'ın ikisi de aynı makine üzerinde bulunabilir. Başka bir makinedeki X serverdan da hizmet alınabilir.

Client üzerinde yapılmaya çalışılan herbir işlem komut olarak sunucuya iletilir. Sunucu görsel bir etkiye sahipse bunu client üzerine yansıtır.

Sanki orada X server yokmuş gibi makine ile client etkileşmede bulunuyormuş gibi etkiye sahiptir.

Window Manager

window sisteminin yönetimiyle ilgilenen yazılımlar. Borderlar, sliderlar içerebilir.

pencere yöneticileri:	kwm	metacity	mwm
	↓	↓	↓
pencereler:	KDE (default)	GNOME (default)	motif standalone

Desktop enviroment

Ör: KDE, Xfce, GNOME

Bunlar default olarak OS kurulumu ile gelmezler. Bunları ayrıca kurmamız gerekir. Dosya yöneticileri (Nautilus), ikonları, panelleri, konfigrasyon araçlarını ve appletleri içerir.

$ moror, X mictre

7) UNIX Shell Enviroments

Shell kullanıcı ile OS arasında bir komut satırı arayüzüdür. Sisteme bağlanınca otomatik olarak sunulur. Komut çevirici olmakla birlikte bir programlama dili olarak düşünülebilir. Bu dil yardımıyla shell scriptler yazarız. Bunlar sıradan bir dosya gibi görülebilir.

Shell Interactivity

- komut satırı üzerinde tetiklemeye yardım eder. Girilen bu komutun optionları ve argumanları belirlemek için parçalara ayırır.

- Kullanıcıya çalışma ortamı sunar.
- Komut satırı üzerinde komut ve argümanları girilirken argümanın otomatik olarak tamamlanma özelliği verir.
- Yazımı uzun zaman alan komutlar için kısa isimler tanımlanabilir.
- Yazılan komutları görmek için yapılandırma yapar.
- Eski komutları tutar, komut satırını düzelme imkanı sunar.

Shell Programming

- Değişkenler
- Kontrol yapıları (döngüler ve şartlar)
- Fonksiyon tanımlama ve çağrımı
- Shell scripler
- Next lecture (ders)

Various Unix Shell

sh (Unix'in orjinal shelli. ilk bu yerleştirilmiş)

ksh (Korn shell)

csh (C shell)

trsh

bash (Bourne again shell) (Linux'un orjinal shelli)

Bourne Again SHell (bash)

Biz bu shell'den yazacağız. Diğer shellerden de bir çok özelliği bize sunar. Superseti sh ile gösterilir. GNU projesinin parçasıdır.

Enviroment Variables

Değişken listesini "env" komutu ile görebiliriz.
Bir değişkenin değerini "echo $varname" komutu ile öğrenebiliriz.
Değişkene değer atamasını C dili syntaxına uygun olarak atarız. $ da kullanırız.

Ör) $HOME → Kullanıcının home değişkenini gösterir.
$PATH → Komutların bulunduğu dizinleri tutar. Girdiğimiz komut bu dizinlerde yoksa komutu koşmaz.
$PS1 → Shellin verdiği komut satırını kontrol eder.
Ör) \u@\h:\~\$
(\u → Kullanıcı ismi, \h → Host name)
$USER → Kullanıcı ismini gösterir.
$HOST → Host ismini gösterir. Bazı makinelerde domain name farklı gösterilir.
$PWD → Çalışma dizininin tam yolunu gösterir.

Setting Enviroment Variables

- "varname = value" ile değişkene değer atanır.
- "PS1 = $USER $HOSTNAME" Orjinal shell prompt'u değiştirir.
- "PS1 = bash_Prompt"
- "PATH = $ PATH: $HOME /bin"
- "PATH = $PATH:~!. do PATH"

• "DATE = 'date'

Komutun ürettiği outputu DATE değişkenine atar.

* (' ') kullanırsak orada ismi geçen komutu koşar.

(' ') veya (" ") kullanırsak içindekini string olarak yazdırır.

Textual Completion

<tab> Aktif komut ya da dosya adını tamamlamaya çalışır.

pu<tab> Alternatifleri verir.

pus<tab> ile pushd <space> geliştirilmiş.

Aliases

"alias shortcut = command" şeklinde her defasında uzun komut ismi yazmamak için kısa isim atarız.

Ör: alias l="ls -F -C"

l → kısa isim

ls -F -C → komutun asıl ismi

Command History

Geçmiş komutları listeler.

"fc -l <m> <n>" önceden yazılmış komutları m'den n'ye doğru listeler.

up ve down cursor tuşlarıyla history listesinde yukarı ve aşağı gidebiliriz.

Editting on the Command Line

bash_bazı satır düzelme komutları sağlar.

Emacs-mode commandları:

- Esc -b → 1 kelime geri gel.
- Esc -f → 1 kelime ileri git.
- Ctrl -a → satır başına git.
- Ctrl -e → satır sonuna git.
- Ctrl -k → kursörden satır sonuna kadar texti yoket.

ksh:
- set -o vi ile vi komutlarını kullanabiliriz,
- set -o vi -tabcomplete TAB ile tamamlama sağlar.

Login Scripts

Bir sisteme her bağlandığımızda aynı yapılandırmaları tekrar yapmamak için bu yapılandırmaların sisteme login olunduğunda yapılmasını sağlar.

Login esnasında çalıştırılan scripler:

- /etc/profile (mutlaka çalıştırılır)
- ~/.bash_profile (çalıştırılır)
- ~/.bash_login (yoksa çalıştırılır)
- ~/.profile (yoksa çalıştırılır)

Loginden sonra çalıştırılan scriptler:

- ~/.bashrc

Logout

- ~/.bash_logout

Example .bash_profile (partial)

- .bash_profile

  umask 022 (0666 & ~022 = 0644 = rw-r--r--)

- .bashrc

  if [-f ~/.bashrc]; then . ~/.bashrc fi

- warm fuzzy environment ile variable setleme.

  → export CVSROOT=~/.cvsroot → export PAGER =/usr/bin/less

  → export EDITOR=/bin/vi

Example .bashrc (bazı kısaltmalar)

- alias bye=logout • alias l='ls -F -C' • alias po=popd
- alias h=history • alias ll='ls -L -l -F' • alias pu=pushd

Login Scripts

- csh için Login shell

  ~/.profile

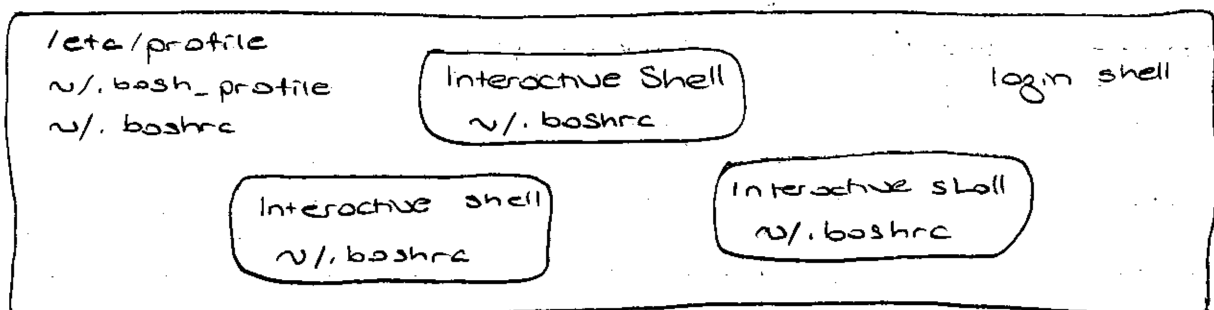
  scriptini icra eder.

- ENV setlenmişse bu dosya her yeni terminal

  için icra edilir.

  → ENV=$HOME/.cshrc → EXPORT ENV (for bash)

Login and Other Shells



(8) Basic Shell Scripting

Shell script komutlardan oluşan bir text dosyasıdır. Bu komut shellin bir komutuymuş gibi icra edilir.

Bu komutlar

- komut satırına yazdığımız herhangi birşeyi
- shell değişkenlerini
- kontrol ifadelerini (if, while, for ...)

içerebilir.

Script Execution

"bash my_script" ile argumentmiş gibi veya "#!/bin/bash" ile shellin ilk komut satırında koşulur.

Simple script

```
#!/bin/bash
echo "Hello, world!"
path = $(pwd)
echo $path
```

Sonuç:

```
Hello, world!
/usr/home/smitt
```

Shell Variables

Değişkenler

- numeric değer,
- string,
- dizi,
- komut satırı argumanları
- fonksiyonlar,
- read-only,

olabilir.

var → variable'yi ifade etsin.

$var

var=100 → değeri 100'e setler

echo "$t = $t" → $t=100' yazar

Numeric Variables

bash reel sayıları desteklemez. Numerik değer olarak sadece integer sayılar vardır. "declare -i var=100" şeklinde deklerasyon ve değer ataması yapılır.

C programlamadaki gibi numeric expressionlar 2 parantez içinde gösterilir.

((exp)) → komutlar için

[exp] → değişkenler için

ÖRN ((var+=1)) or ((var++))

((var2=1+var)) or ((var2=1+$var))

echo $((var*7)) or echo $[var*7]

★C/C++ daki tüm operasyonlar uygulanabilir.

+, -, *, /, %, &, |, <, >, <=, >=, ==, !=, &&, ||

String Variables

" " arasındaki değerler string olarak duşunsülür.

((var2=1+$var))

ise integer olduğunu gösterir.

sonrası koşulmaz.

Bir string üzerinde işlem yapmak için şu syntax kullanılır.

$ (string : n)

Bu komutla stringin ilk n karakterinin atılmasını söyler.

$ (string : (-n))

Bu komutla stringin son n karakterini söyler.

$ (string : n : m)

Bu komutla (n. ile (m+2). (hariç) arasında bulunan karakterleri verir.

$ (#string)

Bu komutla stringin uzunluğunu alırız.

string_var1 = $string_var1 $string_var2

Bununla stringin değerini alıyor. — sondaki 2 string arasında boşluk kalır.

Variable Expansion

${name} → Basit

${name}: -word} → name setlenmemişse wordu kullan.

${name := word} → name setlenmemişse wordu kullan ve ayır

${name : ? [word]} → name setlenmemişse 'word' errorüns u

${name : + word} → name setlenmişse wordu setlenmemişse null kullan.

${name%word} → en küçük son ek paternini çıkart.

${name%%word} → en büyük son ek paternini çıkart.

${name#word} → en küçük ön ek paternini çıkart.

${name##word} → en büyük ön ek paternini çıkart.

Array Variables

~~Array~~ bir değerler listesidir size atanmak ~~zorunda~~ değildir.

- ${name[index]} → bir değeri referanslar.

  ${a[3]} → 4. pozisyondaki değer

  $a → 0. pozisyondaki değer ≡ $a[0]

- declare -a sports → diziyi ifade eder.

  sports = (basketball football soccer)

  sports [3] = hockey

- sports = {(football basketball)}

  moresports = {$ sports tennis}

- ${array[@]}
  ${array[*]} } Dizinin tüm karakterlerine erişir.

- ${#array[*]} → Dizinin boyutunu verir.

- Dizilere komutların çıktılarını da atayabiliriz.

  $ files = ['ls']

  echo ${files[*]}

Exporting Variables

Child süreçler oluşturulduğu zaman parent içerisindeki değerlere child süreçler içinden de erişilebilmesini sağlar. ("export var" ekleyerek)

Command Line Argument

Argumentler scripte ulaşıyorsa herbirine $1, $2, $3, etc. değerleri verilir.

$0 → scriptin adı

$* → $0 hariç birbirinden boşlukla ayrılmış argumanların stringi

$@ → $0 hariç argumanların dizisi

$# → arguman sayısı

Output and Quoting

Standart outputa mesaj göndermek için

echo a message

kullanılır.

$ → Değişkenlerin değerleri

` → Komutların koşumu gerçekleştirilir.

-n → Bir prompt durur.

Ör: echo -n "Yes/No?"

Ör: echo "`date +%D`" # 04/30/05

(+%D: date'in hangi formatta gösterileceği.)

\ escape karakterleri ($, ", ...) kullanmak için.

Return Values

Scriptler integer değer geri döndürebilirler. Bu geri döndürülen değeri $? ile erişebiliriz. Bu değeri geri döndürmek için return N kullanabiliriz. Bunu daha çok koşullu ifadelerde kullanırız. En son durum hakkında bilgi edinmek için uygun

0 → komutun başarıyla koştuğunu gösterir.
↓
returned value

Bu değer aslında bizim koştuğumuz sistem komutu içerisinde bulunan exit komutunun parametresine karşılık gelir.

Shell Variables

3 sınıfa ayırırız:

Kullanıcı tanımlı değişkenler: örnek

Environment değişkenler: PATH, HOME, TERM, PS1, ...

Predefined değişkenler: Formal parametre gibi değişkenler önceden tanımlanmıştır ve bu gruba girer.

* Değişken isimleri alt çizgi içerebilir ve alt çizgiyle başlayabilir.

Ör: FRUIT, TRUST_NO_1, _2_TIMES

* arg="ls -a -l"
↓
Burada boşluk bırakmamalıyız.

Reading User Input

Standart inputu okumak için read kullanılır.

Ör: $ cat read_name

```
echo "Please enter your name and surname:"
read name1 name2
echo "Welcome to CE Dept, KTU, $name1 $name2"
```

Commond and Arithmetic Exponsion

stdout ile komutu yer değiştirmek için

var='commond'

var= $ (commond)

$((expression)) → expression değerini yerdeğiştirmek için

Computation on variables

expr commond

expr val1 op val2 → expr $val1 op $val2

Örn. $>expr 5+7 → 12

$>expr 6-3 → 3

$>expr 3*4 → 12
çarpma için

$>expr 24/3 → 8

$>sum=`expr 5+6` $>echo $sum → 11

$>echo 3*4
↓
çalışma dizinindeki dosyaların adı

$>a=12 $>b=90 $>echo sum is $a+$b → 12+90

Handling Shell Variables

$1-$9 → yerel parametreler

$0 → komut adı

$# → yerel argümanların sayısı

$? → Son koşan komutun çıkış (exit) değeri (0,1,2...)

$$ → Shellin process ID'si

$! → En son koşulan komutun process ID'si

$- → Shell çağrılırken shelle verilmiş optionlar.

$* → Shell'in tüm argumanlarını içeren string

$@ → $* ile aynı.

Notes!

• $*, $@ '' kullanılsa da kullanılmasa da değişmez.

• "$*"

• "$@"

Passing Arguments

Shell scriptlerine komut satırı üzerinden parametreler geçmek için $1 ve $9 arasındaki parametreleri kullanmalıyız. Komut ismi hariç 9 parametre olabilir.

Eğer 9'dan fazla parametre girilmeliyse o zaman shift komutunu kullanabiliriz. Shift komutu argumanları birer birer sola doğru kaydırır. Bir kez shift kullanıldığında $9'un tutacağı değer $10'un değeri olur, $0 atılmış olur.

örn) $ cat pass_arg

Ör: $ cat sum_arg

```
sum=0
while ( $# -gt 0)
do
    sum= `expr $sum + $1`
    shift
done
echo sum is $sum
```

Conditional Exec. Operations

&& → (Ör: command1 && command2) 1. komut başarıyla sonladıysa 2. komut icra edilir.

Bir komutun koşulabilmesi önceki koşuma bağlıysa kullanılır.

|| → 1. komut başarısız olduysa 2. komut icra edilir.

Başarı 0, başarısızlık 1 ile gösterilir.

Ör: ls |grep "mydoc.doc" && rm mydoc.doc

Dosya mevcutsa çalışma dizininden çıkar.

Ör: cat mydoc.doc || echo "file not found"

Dosya mevcut değilse bulunamadı mesajı ver.

Conditional Statements

Koşullu ifadeler ve bir kullanımı $? ile gerçekleştiriliyor.

Conditions

((condition)) → integer kullanım

[[condition]] → string kullanım

örn ((a==10))

((b>=3))

[[$1 =-n]]

[[-e $file]] → Dosya exist mi?

[[-f $file]] → Dosya regular mı?

[[-d $file]] → Dosya directory mi?

[[-L $file]] → Dosya sembolik link mi?

[[-r $file]] → Dosya read özelliğine sahip mi?

[[-w $file]] → Dosya write özelliğine sahip mi?

[[-x $file]] → Dosya koşulabilir mi?

[[-p $file]] → Dosya pipe mı?

The if Statements

```
if test
then
    commands (if conditions is true)
else
    commands (if conditions is false)
fi
```

* else if yok elif var (elif test then command)

test commandı

Komutların başarılı ya da başarısız sonuçlandığını gösterir.

-eq → equal
-ne → not equal
-gt → greater than
-lt → less than
-ge → greater and equal
-le → less and equal

Stringler için:

= → equal
!= → not equal
-z → uzunluğun 0 olup olmadığı
-n → stringin en az 1 karaktere sahip olduğunun kontrolü

Dosyalar için:

-f → Dosya mevcut
-s → Dosya var ve boyutu 0 değil
-d → Directory mevcut
-r → Dosya var ve okunabilir,
-w → Dosya var ve yazılabilir,
-x → Dosya var ve koşulabilir,

Mantiksal operatörler:

- -a → AND
- -o → OR
- ! → NOT

Örn
```
if who|grep -s ahmet > /dev/null
then
        echo ahmet is logged in CS lab
else
        echo ahmet not available in CS lab
fi
```

who → şu anda login durumunda olan kullanıcılar

grep → username arama.

- case expression in pattern1)   commands;;
                     pattern2)   commands;;
                     *)   commands;;
  esac

Örn
```
case $1 in -a) cmds related to option a;;
           -b) cmds related to option b;;
           *) all other options;;
esac
```

Örn
```
clear
echo "1. Date and time"  echo
echo "2. Directory listing"  echo
echo "Enter choice (1 or 2): "
read choice
case $choice in 1) date;;  2) ls -l;;  3) echo error;;
esac
```

For Loops

```
for var [in list]
do
  ...commands
done
```

While Loops

```
while command-list1      → Başarılı olduğu sürece
do
  command-list2
done
```

* break, continue, return de kullanılabilir

Until Loops

```
until command-list1      → başarısız olduğu sürece
do
  command-list2
done
```

* do ---- done
  if ---- fi
  case --- esac
  } ... yerlere komut yazılır

Select Loop

```
select name in word1 ---- wordN
do
  list
done
```

Selectin kullandığı özel çevre değişkenleri var-
dır. $REPLY kullanıcının seçimini verir.

Using basename

Absolute veya relative path üzerinde dosya-
nın ismini ortaya çıkarır.

Ör: $> basename /usr/bin/sh

sh

Using printf

echoya benzer. Ama echo '\n' i otomatikman i-
çerir, printf içermez.

Formatlanmalıdır.

printf format arguments

% [-] m.n x

x → Formatlama dizisinin tipi

m → min dal uzunluğu

n → max dal uzunluğu (tamdeğer olarak kullanılır)

-(-) → Alan içerisinde konumlamada kullanılır. (default sola yanaşık)

- x: stringler için s
  karakterler için c
  integerlar için d
  hexal sayılar için x
  octal sayılar için o
  üstel floating pointler için e
  sabit floating pointler için f
  sıkı floating pointler için g

} olarak kullanılabilir.

ÖN #!/bin/bash

```
printf "%32s %s \n" "Filename" "File Type"
for i in *;
do
  printf "%32s" "$i"
  if [-d "$i"]; then echo "directory"
  elif [-h "$i"]; then echo "symbolic link"
  elif [-f "$i"]; then echo "file"
  else echo "unknown"
  fi;
done
```

* yerine çalışma dizinindeki dosya isimleri gelir.

tee command

Pipelara yazılan ara değerleri bir dosyaya yazmak için kullanılır.

```
$> date | tee now
$> ls | tee list | wc
$> ps |-ael | tee processing | grep
"$UIT"
```

18 satırı 18 words 134 karakteri olduğunu verir.

Formatted Output

Directories: RCS, dev, humor, images, install, java

File: index.html.

Associating Files With a File Descriptor

Unix sistemlerinde herbir koşan sürece 3 standart file descriptor default olarak verilir.

Standart Input (STDIN), 0
Standart Output (STDOUT), 1
Standart Error (STDERR), 2

Biz default olarak bu dosya betimleyicilerin terminal ile ilişkilendirildiğini biliyoruz.
Bir dosyayı yazmak için açmak istediğimizde exec komutunu kullanabiliriz.

exec n> file → dosyaya yaz. (bir satırdan itibaren)
exec n>> file → dosya sonuna ekle.

Ör) $exec 4 > fd4.out

Bu komuttan sonra 4 fd4.out dosyasını belirtmek için kullanılır.

General Input/Output

command 1 > file } output
command 1 >> file
command 0 < file } input

Redirection to seperate files

Ör) command 1> file 1 2> file2

Bu işlemle STDOUT file1'e, STDERR file 2'ye yönlendirilmiş.

Return Value

```
function add_two {
    ((sum=$1+$2))
    return $sum
}
add_two 1 3
echo $?
```

Fonksiyonlar arası data paylaşımını popd, pushd, dirs ile yapabiliriz. Stack'ten direk silme, ekleme ve listeleme yaparlar.

↓ Fonksiyonun başarı ile tamamlanıp tamamlanmadığını belirten değeri geri döndürür.

echo Command

echo [options] [string, variables...]

-n Yazdıktan sonra satır atlamaz.

-e POSIX'tekileri kullanmayı sağlar.

Display colorful text

$>echo "\033[34m Hello!"   Hello! → mavi

\033 → bazı aksiyonlar alan escape karakteri

34 → parametre (mavi renk)

[→ CSI'yı başlatır.

m → actionu belirler.

Genel syntaxı: echo -e "\033[escape-code message"

h ANSI modu setler.

l ANSI modu resetler.

m Renkli ve bold gösterir.

q numlock, capslock...

s cursorun o anki pozisyonunu saklar.

u cursor pozisyonunu eski haline çevirir.

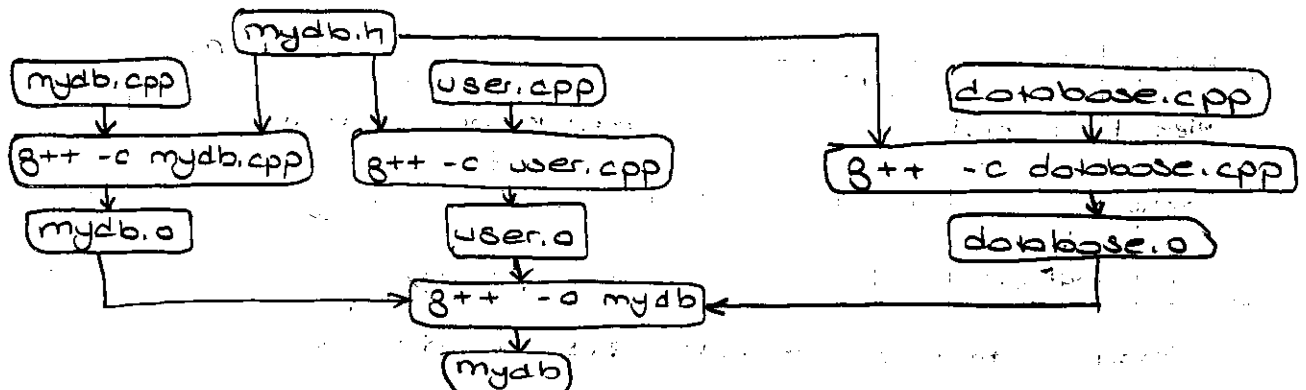
(10) Software Development: g++ and make

Seperate Compilation scenario

source files → mydb.cpp , user.cpp , database.cpp

common headerfile → mydb.h executable → mydb

Build Dependecies



Using make in Compile

Eğer bir kaynak kodun derlenmesi birden çok dosyanın içeriğini değiştiriyorsa make kullanırız.

Basic Operation of make

[mm] akeafile

Eğer derlenecek program başka bir kaynak koda bağımlıysa öncelikle o kaynak kod derlenir.

```
mydb: mydb.o user.o database.o
⇒ g++ -o mydb mydb.o user.o database.o
mydb.o : mydb.cpp mydb.h
⇒ g++ -c mydb.cpp
user.o : user.cpp mydb.h
```

→ g++ -c user.cpp

database.o : database.cpp mydb.o
(target)

→ g++ -c database.cpp

Part of a makefile

Dependence Lines;

- Hedef isimleri ve bağımlılıkları içerir.

- Dosya ve hedef bağımlılıkları

Commands;

- Daima tab karakteriyle başlar. Dependency line'nın altındadır.

Macros and Special Variables

MACRONAME = macro_value → tanımlama

${MACRONAME} → kullanım.

$@ → hedefi temsil eder $? → bağımlılıkları temsil eder.

Invoking make

make → ilk targeti inşa eder

make targets → tüm targetleri inşa eder.

-n komutları koşma, listele   -f <file>: <file>'ı [Mm]akefile yerine kullan

Suffix Rules

${CC} ${CFLAGS} -c $< → file.cpp'yi temsil eder.

* Komutlar # ile başlar.

* Komut satıra sığmazsa '\' ile alt satıra geçilebilir.

Other Useful makefile Tips

- clean:

```
rm -f mydb
rm -f *.o
```

exe ve obj kodlarını silen komutlar.

(11) Debugging

Why use a debugger?

Hata tespitinde kullanılır (Programı satır satır koşulur ya da breakpointler kullanılır).

Debugging with gdb

-g kullanılmalıdır.

Basic gdb commands

Help	help
Breakpoints	break main
Running	run arg-list
Step to next line	next
Step into functions	step
Continue running	cont
List source	list
Quiting	quit

- Running GDB with a core dump
- NetBSD: gdb name name.core
- Solaris: gdb name core

Execution commands

list or l → list

→ list main

→ list 56

run or r → run

→ run file.txt. file2 .txt

next or n → bir sonraki satırı koşma

step or s → bir sonraki satıra geçme

Breakpoint commands

break or b → bir break point setle

break 10   break main

delete or d → bir break pointi silme

delete

delete 2

continue or c → durdurulmuş koşmayı devam ettir.

Program information commands

print or p → değeri yazdır.

print x*y

print function (x)

display → görüntülenen değere devam et.

undisplay → görüntülenen değeri sil.

where → Aktif fonksiyonun stackını göster.

16 04 2007

Miscellaneous Commands

set → değeri değiştir.

    set n=3

help or h → help menüsünü göster

    help
    help step
    help breakpoints

quit or q → gdb'den çık.

I. VİZE SONU

Sınav Çözümleri

1. $> * → "ls" yazar.

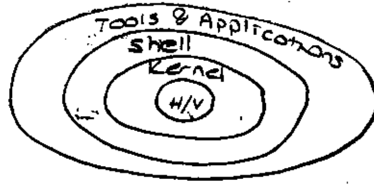
   $> '*' → "aa b cc" yazar.

   $> ? → b scripti koşar. "123" yazar.

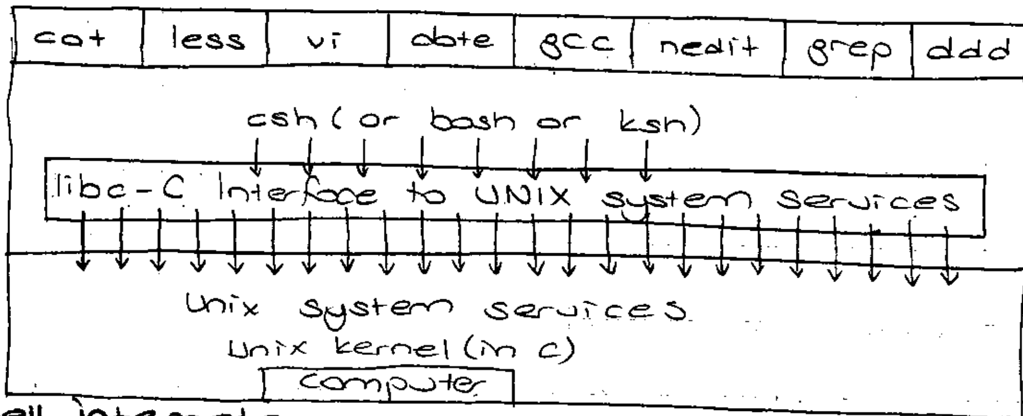
   $> '?' → "bash: 123: command not found" yazar.

2.

(13). File Management



Parts of UNIX OS



Shell intepreter

Shell C'de yazılmış bir programdır. Komut satırına yazdığımız

#! /bin/sh

ile shell yorumlayıcısını koşarız.

Parser tread'ı syntax analizi yapar, komut satırındaki veriyi anlamlı parçalara ayırır ve dizilimini kontrol eder. Parse işleminden sonra programın nasıl icra edileceğini belirler.

Standart Libraries

Standart kütüphaneler içinde sistem çağrıları bulunmaz. Bunlar doğrudan kernel tarafından yürütülen ve OS'ye erişim sağlayan program parçalarıdır.

Library modülleri genellikle birçok obje dosyasından (makina dilindeki koşulamayan, eksik parçalara sahip dosyalar) oluşur. Bir library dosyası üretmek için

ar crv mylib. a *.o

yazarız.

Sistem libraryleri /usr/lib ve /usr/local/lib'de bulunur.

Standart Libraries

.a libraryleri paylaşılmaz, koşulacak exe cod içerisine yazılır. Eğer sistemde aynı libraryi kullanacak birçok komut oluşturulacaksa bu libraryler birçok kez tekrarlandığından ana bellekte şişme olur. Diğer yandan performans artımı sağlar.

.so libraryleri paylaşımlıdır. Ana bellekte sadece 1 kopyası vardır. Tüm süreçler bu kopyayı kullanır. Bellek kullanımını azaltır ve performans düşüşüne sebep olur.

System Calls

libs moduls ile dolaylı yoldan sistem çağrılarını koşarız, libs, sistem çağrılarına C arayüzü sağlar. C kaynak kodu içerisinde library leri kontrol eden sistem çağrılarını çağırabiliriz.

System callar 4 ana gruba ayrılır.

- Dosya yönetimi
- Sistem yönetimi
- Hata ve sinyal yakalama
- Haberleşme

Sistem çağrılarını bize truss gösterir.

System Call Routine

— ASSEMLY ve EXPLANATION SLAYTTAN ÇALIŞ. —

Executing a Program

crt0 özel bir start-up routine'idir. Her zaman programa link edilir. Programa geçen argümanları alır ve main'i çağırır.

libc kütüphanesi otomatik olarak koşulan komuta (c) link edilir. Böylece printf, open gibi komutlar icra edilebiliyor.

C versus C++

C'de stringler karakter dizileri olarak görülür. Deklerasyonlar her komutun başında olmak zorundadır, miras alınamaz.

C++'da

printf yerine cout

scanf ve fgets yerine cin

kullanılır.

Basic File I/O

UNIX'te herşey (printer, terminal, ...) bir dosya olarak tutulur. Bu herbir device'ın kullanımı, veri yazma dosyaya veri yazma gibidir.
Dosyalar okuma ve yazma için açılabilirler.
I/O sistem çağrılarını kullanmak için <stdio.h> dosyasını sistem çağrılarına eklemeliyiz.
Birçok UNIX I/O

- open
- close
- read
- lseek
- write

ile başarılabilir. Herbir sistem çağrısı bir file descriptor (integer) temsil edilir.
3 dosya otomatik olarak açılır.

FD0 : standart input
FD1 : standart output
FD2 : standart error

Bu 0,1,2 değerleri INIT sürecinden gelir. Bu 3 açık dosyayı shell miras alır ve ihtiyacı olan süreçlere verir.
Bir dosya açıldığında en küçük serbest FD'yi alır.
man -s 2 <systemcall> (daha fazla bilgi için)

open()

fd= open(path, flags, mode);

path: char*, absolute or relative path

flags:

O_RDONLY → okumak için aç.

O_WRONLY → yazmak için aç.

O_RDWR → okumak ve yazmak için aç.

O_CREAT → bu dosya yoksa oluştur.

O_TRUNC → dosyanın içeriğini sil.

O_APPEND → dosya sonuna veri ekle.

mode: Bayraklar arasında O_CREAT bulunması.

Geriye döndürülen değer dosya tanımlayıcısı ismi-
ni alır.

fd= open ("myfile", O_CREAT, 00644);

Örn: Bu dosya oluşturulmuştur. Okuma ve yazma ya-
pabilmek için bu bayrakları da eklemeliyiz.

read()

Dosya açılıp bir FD elde edildikten sonra bu FD'yi
diğer sistem çağrıları (Örn: read, write...) için kul-
lanırız.

bytes= read (fd, buffer, count);

fd: FD'nin adı

buffer: count byte arraye pointer

count: okunabilecek max byte sayısı

Bu satır okunan byte sayısını veya -1 (error) döndürür.

Ör) int fd = open ("someFile", O_RDONLY);
    char buffer [4];
    int bytes = read (fd, buffer, 4);

write()

bytes = write (fd, buffer, count);

Buffer içeriği dosyaya yazılır. Geri döndürülen değer yazılan karakter sayısı ya da -1 (error)'dur.

fd, buffer ve count'un anlamları read()'deki ile aynıdır.

```
int fd = open ("someFile", O_WRONLY);
Char buffer [4];
int bytes = write (fd, buffer, 4);
```

close()

return_val = close (fd);

Bu satırla fd dosya tanımlayıcısı serbest bırakılır. Başarılı ise 0, başarısız ise -1 değerini geri döndürür.

* Sadece exit() sistem çağrısı geriye bir değer döndürmez.

lseek()

retval = lseek (fd, offset, whence);

file pointer'ın yerini değiştirir.

offset: byte sayısı

whence:

SEEK_SET → Dosya başlangıcından offset kadar ileri

SEEK_CUR → Mevcut pozisyondan offset kadar ileri

SEEK_END → EOF'dan offset kadar ileri

Dosya başlangıcından offseti veya -1 (error) geri döndürür.

EKSİK >

File I/O using FILES

Unix programı yüksek seviyeli G/Ç fonksiyonları içerir.

fopen()

fread()

fwrite()

fclose()

flseek()

Bunları kullanmak için include <stdio.h> gerekir.

Bunlar FD değil FILE datatipi kullanırlar.

Copying stdin to stdout

```
# define BUFFSIZE 8192
int main (void) {
    int n;
    char buf [BUFFSIZE];
    while ((n= read (STDIN_FILENO, buf, BUFFSIZE )) > 0)
      if (write (STDOUT_FILENO, buf, n) != n)
         printf ("write error");
      if (n<0)
         printf (" read error");
}
```

I/O Efficiency

BUFFSIZE arttikça

kullanicinin CPU kullanimi azalir

sistemin CPU kullanimi azalir

clock zamani azalir

– 〃 azalir

14 05 2004
64

Executing a Program

UNIX sistemlerde programlar exe sistem çağrısı ile icra edilirler. Bunların birkaç türü var. Mevcut adres alanı üzerinde yeni bir program koşulur.
exec başarılıysa geri dönmez, başarısızsa -1 döndürür.

Örn **Program x:**

```
int main () {
    int i=5;
    printf ("%d\n",
    exec
    printf ("%d\n",
}
```

Program y: printf ("hello\n");

exec () properties

exec ile yeni program koşulduğu zaman bu önceki çevreden miras alır. Miras alınan veriler adres alanındaki veriler değildir, sürecin çevresini oluşturan verilerdir. Bu veriler;

- Sürecin PID'leri, UID'leri, GID'leri ve Session ID'leri.
- Sürecin açtığı dosyalar ve FD'ler.
- Kontrol edilen terminaller
- Henüz iletilmemiş sinyal ve alarmlar.

- CWD, root dizini, kaynak limiti.
- dosya kilitleri.
- dosya modu oluşturma maskesi.

exec() versiyonları

```
execl(char *path, char *arg0, ... (char *)0);
execv(char *path, char *argv[]);
execle(char *path, char *arg0, ... (char *)0, char *envp[]);
execve(char *pathname, char *argv[], char *envp[]);
execlp(char *file, char *arg0, ... char ...
execvp(char *file, char *arg[]);
```

Waiting for a Process to Terminal

Bir child süreç oluşturulduğunda child ve parent zaman paylaşımlı ya da eş zamanlı icra edilir. Parent childın icrasını tamamlamasını bekler.

int wait(int * status)

wait'i çağıran süreç tüm childlar koşuyorsa bloklanır, hiç child processe sahip değilse hata-yı geri döndürür, childın koşması sonlandıysa childın termination durumunu geri döndürür.

2. Vize Sonu

28. 05 2007

Signals

Yazılımla gerçekleştirilen bildirimlere verilen isimdir. Bunlar

- floating point hataları
- childın ölümü
- alarm clock çağırıldığında
- control-c
- control-z

Bu sinyaller sistem tarafından da herhangi bir process tarafından da gönderilebilir.

What are signals for?

Sinyaller süreçler arası senkronizasyonun bir biçimini kurmak için kullanılırlar.
İcra sonlanması parent sürece bir child sinyali ile belirtilir.
Processler arası veri iletimi için pipelar ve soketler kullanılır.
Sinyaller makine kesmeleri ve

Software Interrupts

Sinyallere bu adı da verebiliriz. Bunların procedürleri <sys/signal.h> şeklinde listelenmiştir. kill -1 tüm süreçleri sonlandırır.

Signal Table

Burada sinyaller için yapılacak işlemler bulunur;

Sending a signal

Komut satırında kullanım:

kill [-signal] pid [pid] ...

Bu sinyali alan süreçler genellikle sonlanırlar.

Signalling between processes

kill (int pid, int sig);

Kill Usage

retval = kill (pid, signal)

- → pid > 0 ⇒ Bu processe
- → pid = 0 ⇒ Bu gruptaki processlere
- → pid = -1 ⇒ Tüm processlere

} gönderilir.

- root ⇒ Tüm sistem processlere
- !root ⇒ Aynı uid'deki processlere

What Happens at "signal time"?

kill sinyalini alan süreç icrasını sonlandırmak, stop sinyalini alan ise durdurmak zorunda. Sinyali reddedemez.

Signal Handling

void (*signal (int sig, void (*func) (int));) ()

Sistem çağrısıyla ilgilenmek için bu fonksiyon kullanılır. Burada func SIG_DEL veya SIG_IGN olabilir.

Timer Signals

SGLALRM

SGUTALRM

SGP.

Advenced Signal Interface

Sinyollerle ilgilenmek için daha gelişmiş bir ara-

raydedır.

#include